

And away we spoof!!!

Table of Contents

<u>And away we spoof!!!</u>	1
<u>Notes on stopping arpspoof, the program</u>	1
<u>Dsniff utilities</u>	1
<u>Bandwidth Control</u>	3
<u>Bandwidth usage</u>	5
<u>MRTG</u>	5
<u>Interpreting MRTG</u>	5
<u>IP Flow Meter (ipfm)</u>	6
<u>Interpreting ipfm output</u>	7
<u>IPTraf</u>	8
<u>Berkeley Packet Filter (bpf) Quickie</u>	9
<u>Tcpdump</u>	9
<u>Interpreting tcpdump traffic</u>	10
<u>NTOP</u>	11
<u>Conclusion</u>	12
<u>Defenses</u>	13
<u>Read Carefully!</u>	13
<u>The Heart of the monitoring</u>	15
<u>Essential preparation</u>	17
<u>Software Used</u>	18
<u>Ripped from the Headlines</u>	20
<u>Ngrep</u>	21
<u>Snort</u>	21
<u>Security Considerations</u>	23
<u>Data Security</u>	24
<u>Remote Access</u>	25
<u>Restricting PAM-style</u>	27
<u>The chosen are few</u>	28
<u>Hand in the googie jar</u>	28
<u>Other considerations</u>	29
<u>Notes</u>	31
<u>Thanks</u>	32
<u>'To spoof or not to spoof, that is the packet!</u>	32
<u>Dsniff 'n the Mirror</u>	33

And away we spoof!!!

Table of Contents

<u>Introduction</u>	34
<u>Why?</u>	34
<u>TODQ</u>	35

And away we spoof!!!

If you are using port mirroring then you only need to read the section on enabling ip forwarding. Not enabling it won't have an affect on the network but you may not be able to see any traffic depending on the OS.

By default **IP Forwarding** is disabled on Linux. Before Eve can start forwarding traffic to **Trent**, IP Forwarding must be enabled on the computer running arpspoof, hereby called **Eve**. This is done with the command:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

However, this is enabled in a file named `/etc/sysctl.conf`

```
net.ipv4.ip_forward = 1
```

so IP Forwarding will always be enabled if the network interfaces are reset on **Eve** or if the machine is restarted. (**NOTE: If IP Forwarding is not enabled and arpspoof is running the network it is running on will come to a stand still. Be sure you have IP Forwarding enabled!!!**)

"**Arpspoof**" at the core of the monitoring can be run by using the following command:

```
/usr/sbin/arpspoof 192.168.0.1 2>/dev/null 1>/dev/null &
```

"**2>/dev/null 1>/dev/null**" is used to keep the output from being redirected to the console (It is sent into nothingness, a little Eastern Philosophy humor). "**&**" is used to put the process in the background. That is really all there is to it to take over a network, see why it has the negative publicity. All traffic intended to go to **Trent** will first be redirected to **Eve** and then to **Trent**.

Notes on stopping arpspoof, the program.

Arpspoof should be stopped with a graceful shutdown, else the network it is running may become incapacitated. Also, don't pull the network cable out of Eve while arpspoof is running or else it could incapacitate the network. To stop arpspoof gracefully, hold your mouth at a right angle and type:

```
/bin/kill -15 `sbin/pidof arpspoof`
```

"**-15**" is a graceful shutdown. This will cause arpspoof to send gratuitous arps back to Trent and network operations will continue. If it is stopped abruptly don't fret. Just restart the network interfaces on Trent. That is all. You'll also want to put a kill statement in your shutdown scripts so that it will stop gracefully if you need to reboot Eve and forget to stop arpspoof. In the `/etc/rc.d/rc6.d/00killall` script add the line:

```
/bin/kill -15 `sbin/pidof arpspoof`
```

This will return the network back over to Trent. Similarly, you'll want to add it to your network startup scripts so in your `/etc/rc.d/rc.local` file add the line:

```
/usr/sbin/arpspoof 192.168.0.1 2>/dev/null 1>/dev/null &
```

Don't forget the ampersand (**&**) on the end, else you won't get a login prompt.

Dsniff utilities

A few other other utilities that come with dsniff of use can be effective for bandwidth control, "**tcpkill**", "**tcpnice**", and "**urlsnarf**". See Chris Prosis's and Saumil Udayan Shah's article on using urlsnarf ["Catching a](#)

And away we spoof!!!

[Hacker in the Act](#)" Dug Song and I aren't the only ones who find good uses for Dsniff. If you have a court order to monitor someones browsing you can use **webspy**. That's right a **COURT ORDER!!!!**

Start up your web browser. Then run the following commands (thanks Stephen):

```
Trent: 192.168.0.1
Eve box 192.168.0.2
User you have a court order to monitor: 192.168.0.3
```

```
/usr/sbin/arp spoof -i eth0 -t 192.168.0.3 192.168.0.1
```

Tricks the user's machine (the one you have to court order for) to go to Eve instead of the real gateway.

```
/usr/sbin/arp spoof -i eth0 -t 192.168.0.1 192.168.0.3
```

This tricks Trent into sending traffic to Eve first, and then Eve will send it to the user you have a court order to monitor.

Why all that? Because your machine is going to assume the ip address of another machine on the network. The user you have a court order against will get that message that *'his/her ip address is in use on the network'*. The above commands will prevent that. In a few moments you should start seeing the same sites the user you have a court order for is seeing.

[<<< Previous](#)

Defenses

[Home](#)

Dsniff 'n the Mirror

[Next >>>](#)

Bandwidth Control

[<<< Previous](#)

[Next >>>](#)

Bandwidth Control

Tomasz Chmielewski's [Bandwidth-Limiting-HOWTO](#) is an excellent reference document for CBQ (Class Based Queueing). This will allow you to allocate bandwidth to particular network services. Please refer to the section on CBQ in his howto.

Tcpkill can be used to kill connections to or from a particular host, network, port, or combination of all. These programs take standard bpf (Berkeley Packet Filters) filters so read the tcpdump man pages for examples on how to create those filters. Also, there are other sites on the Internet that have example filters. This can be used for both port mirroring and arpspoofing on Eve. For example, to prevent any connections to the host www.playboy.com use this command:

```
/usr/sbin/tcpkill -9 host www.playboy.com
```

The computer that is attempting to go to that site will be blocked from that site only, but can surf any other site. It is a good idea to either redirect the output into nothingness (`> 2>/dev/null 1>/dev/null`) or into a file for later analysis (`> file.tcpkill`). By default, it will redirect output to the console. More hosts can be specified with the command:

```
/usr/sbin/tcpkill -9 host www.playboy.com and host www.real.com
```

or

```
/usr/sbin/tcpkill -9 host www.playboy.com or ( www.real.com or www.penthouse.com )
```

to block well-known ports eg., napster (**port 8888 and port 6699**) or gnutella (**port 6346**), the command:

```
/usr/sbin/tcpkill -9 port 8888 and port 6699
```

or

```
/usr/sbin/tcpkill -9 port 6346
```

will do the trick.

"Tcpnice" is similar except that it only slows down the connection. The range is 1 through 20. 20 being the slowest. For example, to slow down napster and gnutella traffic to a crawl this command will do that:

```
/usr/sbin/tcpnice -n 20 port 8888 and port 6699
```

or

```
usr/sbin/tcpnice -n 20 port 6346
```

If a particular subnet (**192.168.1.0/24**) was using up a lot of bandwidth this command will slow down that subnet:

```
/usr/sbin/tcpnice -n 20 net 192.168.1
```

This command is a bit drastic because there connection will be crawling. The default is "**-n 10**".

Substituting the tcpkill command can be used to block that entire subnet for a while:

```
/usr/sbin/tcpkill -9 net 192.168.1
```

These examples should get you started.

There are other utilities with dsniff but they don't have any immediate use for this paper. They can either be deleted or made non-executable:

And away we spoof!!!

```
/bin/chmod 000 /usr/sbin/macof
```

(Just repeat for each unwanted binary)

If you are using arpspoof on Eve then you can use the firewall programs ipchains, iptables, or whatever firewall you use to block ports or sites and log them. Then have some firewall logging utility parse the file and have it emailed to you.

[<<< Previous](#)

[Home](#)

[Next >>>](#)

And away we spoof!!!

Bandwidth usage

Dsniff 'n the Mirror

[<<< Previous](#)

[Next >>>](#)

Bandwidth usage

The programs being used to monitor the network are MRTG, IPFM, IPTraf, Snort, darkstat, and NTOP.

MRTG

NOTE!! WITH THE LATEST SNMP VULNERABILITIES, BE SURE YOU HAVE THE PATCHES BY YOUR VENDOR AND CHANGE THE COMMUNITY STRING. "public" is only used in the following examples for demonstration.

MRTG is used to monitor bandwidth throughout the day and it graphically archives the data over a period of a day, week, month, and year. This data can be backed up each week and stored to either removable media or on a remote server. The main config file is `/etc/mrtg.cfg`. The auto-configuration command (`/usr/local/mrtg-2/bin/cfgmaker`) used to generate the config file is:

```
/usr/local/mrtg-2/bin/cfgmaker --global "WorkDir: /usr/local/apache/htdocs/stats/mrtg" --global "
```

This command puts the graphs and logs used to create the graphs in `"/usr/local/apache/htdocs/stats/mrtg"`. The root directory of the web server is `/usr/local/apache/htdocs/` so the directories `/stats/mrtg` will have to be created:

```
/bin/mkdir -p /usr/local/apache/htdocs/stats/mrtg
```

The `-p` switch creates subdirectories within a new directory without having to create each directory separately like this:

```
/bin/mkdir /usr/local/apache/htdocs/stats/ ; /bin/mkdir /usr/local/apache/htdocs/stats/mrtg
```

The date is displayed in bits (`--global "options[_]: bits`). No name lookups are performed (`--noreversedns`), the autoconfigured file is saved to `/etc/mrtg.cfg` (`--output`) and the snmp community name (**public**) and server name to monitor is specified `"public@Eve"`. This command only had to be run once. If the server is destroyed or the hard drive dies then this is the command to run on a fresh install of MRTG.

A cron entry runs the statistics gathering every 5 (`*/5`) minutes; thus, the graph is updated every 5 minutes. Here is the cron entry:

```
*/5 * * * * /usr/local/mrtg-2/bin/mrtg /etc/mrtg.cfg
```

Graphs can then be viewed over the web by browsing to the location specified in the MRTG command above (`/usr/local/apache/htdocs/stats/mrtg`). Since the root directory of the web server is `/usr/local/apache/htdocs/`, the last two directories `"/stats/mrtg/"` is what is specified in the url location bar. Open a browser and type the location to the graphs:

```
http://localhost/stats/mrtg/
```

Click on the **"Eve.html"** link of the name of the server and the graphs should then be displayed. At this point that page can be bookmarked or an initial Monitoring webpage can be created with the absolute path to the "eve.html" file.

Interpreting MRTG

MRTG graphs show the max output for the time periods displayed. Time is in military time. As a quick reminder, if the time is 13 to 23 subtract 12 from that number to get the standard time. So 16:00 would be 4:00 because 16-12 is 4. Each line on the y-axis represents one hour. High spikes indicate a large amount of

And away we spoof!!!

traffic being sent or received.

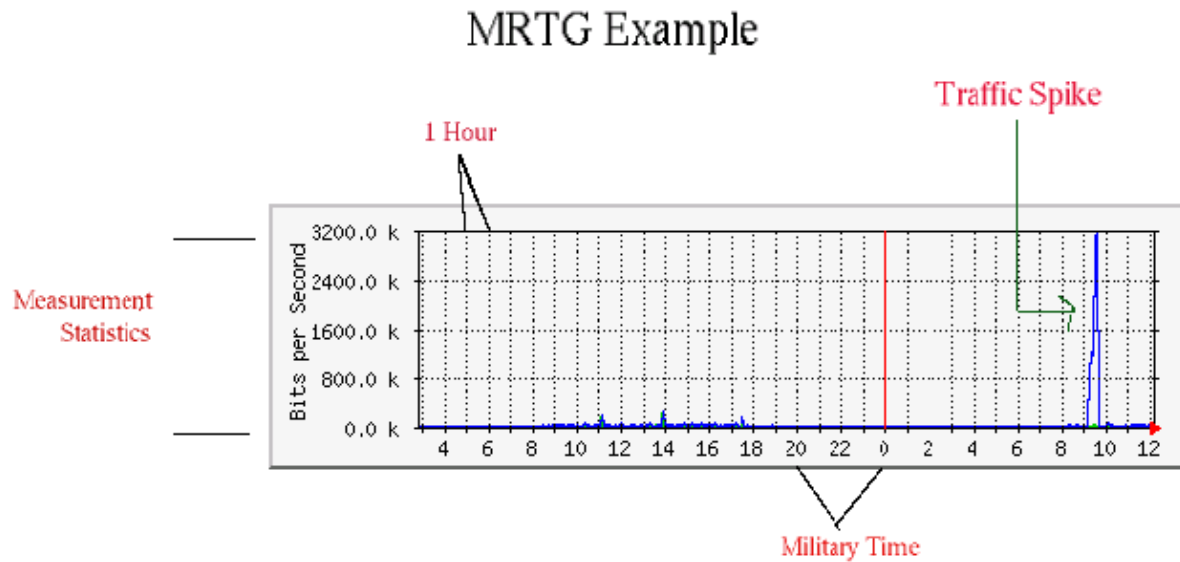


Figure 1.

The example above shows a high traffic spike. Notice how it distorts the normal looking traffic just above the x-axis. Though accurate it is difficult to see those results. That is why we are using multiple monitoring utilities. The daily chart on MRTG shows a two-day period so the graph will return back to normal after that time.

IP Flow Meter (ipfm)

Ipfm collects the amount of data used for each host on the network. It is run in the background with the following command:

```
/usr/sbin/ipfm -c /etc/ipfm.conf
```

(-c) specifies the config file to use. Here's the config file `/etc/ipfm.conf`:

```
# Global variables

DEVICE eth0
# local variables
##### FIRST LOGGING CONFIGURATION #####
#log our subnets 192.168.
LOG 192.168.0.0/255.255.0.0
#Path and name of log file.
FILENAME /var/log/ipfm/ipfm-%A-%d-%B-%Y-%H.%M.%S
#Time to output collected statistics.
TIME 30 minute
#Data is sorted by the host that has sent the most traffic
SORT OUT
#Whether or not to resolve hostnames
RESOLVE

##### SECOND LOGGING CONFIGURATION #####
#Create a new log file.
NEWLOG
```

And away we spoof!!!

```
#Log our subnets
LOG 192.168.0.0/255.255.0.0
#Path and name of log file.
FILENAME /var/log/ipfm/ipfm-week-%A-%d-%B-%Y-%H.%M.%S
# Log on a period of one week
TIME 7 day
SORT OUT
RESOLVE
```

The configuration "**192.168.0.0/255.255.0.0**" is used because 192.168.0.1 is our gateway and it is assumed all of our networks, 192.168.0.0/16, are using it as the gateway as well.

The output of each log file includes the IP (or Hostname), In, Out, and Total traffic for each host. Traffic is in bytes. The example below, in the section **Interpreting ipfm output** is the format of the ipfm-* log file, without the color and bold. Data is saved in the /var/log/ipfm directory so a symbolic link was created to the web directory /usr/local/apache/htdocs/ipfm so data is readily available. The command to do that was:

```
/bin/ln -s /var/log/ipfm /usr/local/apache/htdocs/ipfm
```

Important Security Note: Be very careful when you create symbolic links to your web directory. It is possible that improper server configurations could lead to a malicious person reading very sensitive files by exploiting this link. Read your webserver documentation carefully about creating symbolic links.

All this data can be backed up to the "/var/log/ipfm/backup" directory and viewed from the webserver. First create a new directory under the "/var/log/ipfm" directory named "backup".

```
/bin/mkdir /var/log/ipfm/backup
```

Here is a shell script to perform the backup:

```
#!/bin/sh

#Create a new directory under /var/log/ipfm/backup with "ipfm-" and the current date as the name.
/bin/mkdir /var/log/ipfm/backup/ipfm-`date '+%A-%d-%B-%Y'`

#mv the ipfm daily stats into /var/log/ipfm/backup/ipfm-"CURRENT date"
/bin/mv /var/log/ipfm/ipfm-* /var/log/ipfm/backup/ipfm-`date '+%A-%d-%B-%Y'`;
```

Add the contents to a file under the **/etc/** directory named ipfm-backup.sh. Make it executable only by root:

```
/bin/chmod 700 /etc/ipfm-backup.sh
```

Backups are kept in the /var/log/ipfm/backup/ipfm-"current date" directory, where "**current date**" is the day of the ipfm log statistics, respectively. The cronjob is run at 11:56 each night. Here is the cron entry:

```
56 11 * * * /bin/sh /etc/ipfm-backup.sh
```

The second part of the **/etc/ipfm.conf** file specifies that a log is created once a week with the overall stats for each host.

Interpreting ipfm output

Data output is self-explanatory, except to know that the data is sorted by the host that has sent the most data. That is specified in the **/etc/ipfm.conf** file with the option:

SORT OUT

And away we spoof!!!

Looking at the graph above we can see that the time period the spike occurred was between 9:00 and 10:00am in the morning. All that needs to be done now is to browse to the `/var/log/ipfm/` directory and look for the output files with the appropriate time stamp. In this case the files `ipfm-Thursday-10-May-09.01.53` and `ipfm-Thursday-10-May-09.31.53` will show what host(s) caused the spike. (Note the file name in the previous sentence). The filename comes from the `/etc/ipfm.conf` config file from above. The `FILENNAME` option is `"ipfm-%A-%d-%B-%Y-%H.%M.%S"`. In this example, the file name is `ipfm-Thursday-10-May-09.01.53`. All files will be created in that format.

Snipped for brevity.

Host	IN	OUT	TOTAL
Eve	5006296	306296058	311302354
192.168.0.5	305264283	5483840	310748123
Trent	1538863	11686	1550549

Host **192.168.0.5** is responsible for the spike in traffic, from the "MRTG Example" illustration above. It received over 300 megs of data (**IN**). Which is accurate considering I transferred over 300 megs of data from **Eve** (**OUT**) to test it out.

IPTraff

Iptraf is a superb utility for network monitoring. It allows you to view and log all incoming and outgoing requests to the Internet and on the local LAN. It is a very flexible and lightweight program. It has a very easy to understand interface. IPTraf's website has superb [documentation](#), and screenshots on the website so details will be spared and only mention what we are currently using. Also, it comes with a precompiled ready to run binary. If you want to compile it yourself then you can do that as well. It is run in interactive mode from the console by issuing the command:

```
/usr/local/bin/iptraf
```

IPTraff gives detailed information and statistical data on each host, port, protocol, etc., on **ON-2**. The option to log data will always be prompted before the data analysis is begun while in interactive mode. Hit "Enter" to enable logging or "Ctrl+x" to disable logging. All data is stored in the directory `/var/log/iptraf/` and has a corresponding file for what is being monitored. IPTraf runs in the background (**-B**) with the following command:

```
/usr/local/bin/iptraf -d eth0 -B
```

This gives detailed (**-d**) information on the protocols used and an overall rate of how much bandwidth is used since the first instance of IPTraf was run. The data is updated every 5 minutes and redirected to a file in the web directory on **Eve**. Currently it runs continuously, however that can be changed by adding a cron entry to stop and start it at a certain time of day. The data is continuously updated and averages are kept since iptraf first ran so stopping it each day and starting it again gives a better view of what's going on during that day. The documentation located on the website has more extensive information on what IPTraf can do. However, experimenting with it without the documentation has no affect at all on the network. Only the utilities used with dsniff can alter the network.

A cron entry redirects the output of the 5 minute statistics logging to a file in the web directory:

```
/usr/bin/tail -30 /var/log/iptraf/iface_stats_detailed-eth0.log \
```

And away we spoof!!!

```
> /usr/local/apache/htdocs/iptraf/fiveminutes.txt
```

tail -30 is used because the last 30 lines of the output gives all the information for the last [five minutes](#). And the entire **iface_stats_detailed-eth0.log** file is redirected to the web directory each hour for analysis over a period of time:

```
/bin/cat /var/log/iptraf/iface_stats_detailed-eth0.log > /usr/local/apache/htdocs/iptraf/all.txt
```

Berkeley Packet Filter (bpf) Quickie

Before getting into the other tools, this is a quick primer for the **bpf** filters. **BPF** allows you to select particular packets from the network for viewing. Below are some examples just to get you started. The `tcpdump` man pages have more detailed examples and explanations.

Dump http traffic and don't resolve host names "**-n**":

```
tcpdump -n port 80
```

Instead of a port, the name associated with that port can be used, as long as it is listed in the `/etc/services` file; thus,:

```
tcpdump -n port http
```

would work as long as "`http`" is located in `/etc/services`.

All traffic except "`!`" `http` traffic.

```
tcpdump -n ! port http
```

or

```
tcpdump -n not port http
```

All "`http`" traffic and "`ssh`":

```
tcpdump -n port 80 and port ssh
```

Traffic from host "`192.168.1.7`":

```
tcpdump -n host 192.168.1.7
```

Traffic on network "`192.168.1.0/24`":

```
tcpdump -n net 192.168.1
```

Traffic destined for host "`192.168.1.7's`" port 80:

```
tcpdump -n host 192.168.1.7 and port 80
```

Tcpdump

Tcpdump (WinDump is available for Windows platforms) is used to capture traffic on an interface with the specific purpose of helping to debug network problems. It is also great for capturing data to send to an ISP if problems appear to be occurring on their network, eg. their servers killing connections. We have used tcpdump to debug a lot of problems like that. The tcpdump output can be stored in any directory, provided the filesystem where the directory is located has enough space. Space is crucial as tcpdump files can grow very large. For this example, we'll use the `/log` directory. Here is the command used to capture data:

```
/usr/sbin/tcpdump -w /log/dump_`date +%A-%d-%B-%Y` ! broadcast and !  
multicast
```

And away we spoof!!!

The data collected from the `tcpdump -w` command is stored as a binary file in the libpcap format. This format type is good, as you will see because other programs such as `ngrep`, `snort`, `tcptrace`, `tcprelay`, and other programs use the libpcap format of `tcpdump` to analyze data. Normal viewing yields a bunch of garbage. The ``date +%A-%d-%B-%Y`` command grabs the current Month, Day, and numerical day and Year, respectively, and uses that as the file name. **! broadcast and ! multicast** is used to not log broadcast and multicast as that could cause the file to grow insanely large. These expressions are part of the Berkeley Packet Filter libraries. This allows you to exclude and include data capturing capabilities with data over a network. The `tcpdump` man pages have a lot of information on how to create expressions. I'll include some examples however.

`Tcpdump` can be started from a cron job. Here is a shell script to do that, named **tcpdump.sh**

```
#!/bin/sh
/usr/sbin/tcpdump -w /log/dump_`date +%A-%d-%B-%Y` ! broadcast and ! multicast
Add that file to your /etc directory and make it executable:
```

/bin/mv tcpdump.sh /etc ; /bin/chmod 700 /etc/tcpdump.sh.

Here is the cron entry to start it at 24:00:

```
0 0 * * * /bin/sh /etc/tcpdump.sh
```

`Tcpdump` is stopped at 23:59, moved to a backup directory, and tarred and gzipped with the following shell script named "**tcpdump-backup.sh**":

```
#!/bin/sh

#Grab the pid for the current running tcpdump process
tcpdump_pid=`/sbin/pidof tcpdump`

#kill current tcpdump process
/bin/kill -9 $tcpdump_pid ;

#Move to the backup directory
/bin/mv /log/dump_`date +%A-%d-%B-%Y` /log/backup/ ;

#Tar and gzip the days dump file.
/bin/tar -czf /log/backup/dump_`date +%A-%d-%B-%Y`

Make the file "tcpdump-backup.sh executable and place it in the /etc directory.
```

```
/bin/mv tcpdump-backup.sh /etc ; /bin/chmod 700 /etc/tcpdump-backup.sh
```

Here is the cron entry:

```
59 11 * * * /bin/sh /etc/tcpdump-backup.sh
```

Interpreting tcpdump traffic

For a more thorough explanation of how to interpret `tcpdump` output read [Karen Frederick's](#) excellent three-part articles. She explains the output using `WinDump` but it applies to Unix's `tcpdump`, as well. Richard Bejtlich has an excellent article on [interpreting tcpdump output](#). Also, Stephen Nortcutt's books *Network Intrusion Detection: An Analysts Handbook* [Edition 1](#) and [Edition 2](#) are highly recommended.

NTOP

NTOP is so awesome. There are so many features in this program. It now supports just reading from a libpcap file and not collecting data from the network when it is finished. You've got to take a look at this program. It should be used in moderation as it is still intensive to run with live data capture especially with large networks. I prefer to run it on a libpcap file. It also saves the data in its own database format so it can be referenced for later analysis. My tears of happiness are just flowing. It is an excellent program (sniffle!) and gives superb statistics in a nice web-based format. One second please! (sniffle, sniffle, sniffle!!) This is a cool program.

Ntop has features similar to MRTG and IPTraf combined. Ntop gives graphical statistics in a web-based display. It gives the overall network usage as well as per host usage, the number of packets sent and received by each host, and also statistics from each remote hosts to name just a few features. It also keeps a history of what websites each host has visited.

Ntop was running with the command:

```
/usr/sbin/ntop -i eth0 -f /log/dump_`date +%A-%d-%B-%Y` -F "OurNetwork='net 192'" -r 400 -l 300
```

Ntop reads the tcpdump output file (`-f /log/dump_`date +%A-%d-%B-%Y``) to gather statistics and other info. This is a great feature because it gathers everything that each host has done, how much bandwidth was used, the number of packets each host sent and received, each host's history, pretty graphs, protocols used, it can create matrixes between networks using the filters commands, etc., but it used too much CPU because of the large log file. The filter (`-F`) OurNetwork is created to provide statistics about communication on **ON-2 (net 192)**. Other filters can be created as needed (**see the ntop man pages**). The information collected by ntop is updated every 5 minutes (`-l 300`) and the web based screen is updated every 6 minutes (`-r 400`)

All the information gathered is self-explanatory as the links on the left side of the screen explain what data is to be shown. On the top right you can choose the category of data you want to view.

[<<< Previous](#)

[Home](#)

[Next >>>](#)

Bandwidth Control

Ripped from the Headlines

Dsniff 'n the Mirror

[<<< Previous](#)

[Next >>>](#)

Conclusion

Hopefully, this paper gives some general ideas on what can be done using dsniff and/or port mirroring as an aid in network monitoring. There are many tools out there that can be used to assist with other aspects of auditing a network too. A search on [Google](#), [Securityfocus](#), [Freshmeat](#) (Linux newbies, it is not what the name implies), [LinuxApps](#), and various other websites will yield many other programs that will help you with monitoring. There are other ways to monitor a network but because of our situation this way was inexpensive. The only financial cost was an available Dell Pentium III 600, 6 gig, 256mb RAM server. All the utilities in this paper are free. Also, the floppy distribution, [Trinux](#), contains many of the tools used in this paper. Trinux is the floppy Linux distribution for Network Administrators and Security Administrators. It has many programs that comes with it and the most basic distribution requires only 3 floppies. Everything mentioned in this paper was also done on a 486sx, 16mb, computer running only in RAM with no performance degradation. Trinux is designed to be a portable distro to help Administrators audit and monitor their network. It can be installed on a hard drive as well. We ran Trinux in RAM and mounted the hard drive and redirected all output on the hard drive, then used the SSH utility scp to send it to a more powerful machine for analysis.

It is hoped you find this article useful and I would like to hear how you are using it.

[<<< Previous](#)

Security Considerations

[Home](#)

Dsniff 'n the Mirror

[Next >>>](#)

TODO

[<<< Previous](#)

[Next >>>](#)

Defenses

You should know how to defend against possible malicious use of dsniff and related programs than have to read this paper and wait to the end. By default you can only sniff on the local subnet your are on.. If multiple networks are sharing the same gateway and improper filtering rules were in place then this would work on every network sharing that gateway. Also, hard-coding the mac address of the gateway on the switch would help prevent arp spoofing. That is a temporary fix because a "mac flood" attack can be performed on the switch. A "mac flood" is when a bunch of bogus mac addresses fills up the memory of the switch and could possibly cause it to "fail-open" (yes dsniff has that utility as well called, "macof"). This is essentially the state of a non-switched network where packets are broadcasted to every machine on the network until it finds the intended host. In this state, and on a non-switched network, users only need to put their interface in promiscuous mode to sniff traffic. Also, the tedious task of hard-coding the mac addresses of the network card on each machine can be done. For example, on linux machines, adding the mac address for each machine in the "/etc/ethers" file will prevent arp request and replies from being sent and received. A utility named "arpwatch" can be used to email the administrator if mac address mismatches are detected on the network.

Another sign that arpspoof or similar programs are running is the output of ping. Here is a normal ping:

```
# ping www.google.com
PING www.google.com (216.239.33.101) from 192.168.0.9 : 56(84) bytes of data.
64 bytes from 216.239.33.101: icmp_seq=0 ttl=239 time=50.643 msec
64 bytes from 216.239.33.101: icmp_seq=1 ttl=239 time=59.950 msec
64 bytes from 216.239.33.101: icmp_seq=2 ttl=239 time=49.956 msec
64 bytes from 216.239.33.101: icmp_seq=3 ttl=239 time=49.954 msec
^C
```

Here is a ping with arpspoof running on a machine on the network:

```
PING www.google.com (216.239.33.101) from 192.168.0.9 : 56(84) bytes of data.
From 192.168.0.2: Redirect Host(New nexthop: 192.168.0.1)
64 bytes from 216.239.33.101: icmp_seq=0 ttl=239 time=51.521 msec
From 192.168.0.2: Redirect Host(New nexthop: 192.168.0.1)
64 bytes from 216.239.33.101: icmp_seq=1 ttl=239 time=49.950 msec
From 192.168.0.2: Redirect Host(New nexthop: 192.168.0.1)
64 bytes from 216.239.33.101: icmp_seq=2 ttl=239 time=49.954 msec
From 192.168.0.2: Redirect Host(New nexthop: 192.168.0.1)
64 bytes from 216.239.33.101: icmp_seq=3 ttl=239 time=49.956 msec
^C
```

Notice the line:

```
From 192.168.0.2: Redirect Host(New nexthop: 192.168.0.1)
```

It shouldn't be there. No redirects should be occurring at all. Here it can clearly be seen that all traffic is first sent to Eve (192.168.0.2) then sent to Trent (192.168.0.1).

Read Carefully!

The Dsniff utilities below can affect the entire network. Please be careful when using them!!!! The most important thing to be done is enable IP Forwarding else the entire network, will be incapacitated. Again, ENABLE IP FORWARDING BEFORE RUNNING ARSPOOF.

The utilities used with dsniff can be used to intercept passwords, email, instant message conversations, and other potentially critical information. These tools should be used with the utmost care and only

And away we spoof!!!

authorized users should have access to the server that is doing the monitoring. This is the time to pull out that dusty security book and implement strict access controls and read up more on security.

[<<< Previous](#)

[Home](#)

[Next >>>](#)

'To spoof or not to spoof, that is the packet'

And away we spoof!!!

Dsniff 'n the Mirror

[<<< Previous](#)

[Next >>>](#)

The Heart of the monitoring

Two methods are discussed for monitoring networks, dsniff and "**port mirroring**". Dsniff (<http://www.monkey.org/~dugsong/dsniff/>) is a freely available suite of utilities that comes with tools to assist with network monitoring and auditing. It has received a lot of negative press (see dsniff's web page on "[Recent press:](#)" articles. because it is freely-available and can be run from anyone's computer on a network to sniff traffic, intercept SSL connections, hijack a network, etc.. In addition, it is available on some popular platforms, Windows, Unix, Solaris, and MacOS X. Despite the potential malicious use, this program has more positive (as seen below) than negative use.

Most switches have a feature called "**port mirroring**". Check your switch documentation. This is what should be used if the feature is available on your switch. Port mirroring will achieve similar results as using arpspoof in this paper. Except, with port mirroring you aren't poisoning the cache of the machines on the network. Port mirroring uses a port on the switch which forwards traffic destined for one port to another port. Usually the port for the gateway or the port for the main router is mirrored. This is often used by organizations to setup an Intrusion Detection System. The network interface on the server connected to the port that is mirroring doesn't need an ip address so it is virtually invisible to the rest of the network. IP forwarding may need to be enabled on Eve, check your server documentation. On some switches port mirroring can be setup to mirror one-way or bi-directional traffic. For the best results, bi-directional mirror should be enabled. Also, it doesn't affect bandwidth like arp spoofing.

Arp spoofing adds an extra hop for the traffic going in and out of the network and the icmp-redirects generated also can consume bandwidth on large networks. The redirects that are generated are a result of the monitoring server. It is telling you that a shorter route exist to the gateway than the one that is going to, in this case Eve causes the packets to take another route to the real gateway. In this paper, traffic is flowing out like this:

```
HOST ---> EVE ---> TRENT ---> INTERNET
instead of the faster route
```

```
HOST ---> TRENT---> INTERNET
```

These redirects can be disabled on the monitoring server with the option:

```
echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects
or
```

in the `/etc/sysctl.conf` file add:

```
net.ipv4.conf.all.accept_redirects = 0
```

Then run:

```
/sbin/sysctl -w
```

This will immediately add the new security setting. The `(-w)` switch changes the setting. To be sure it is enabled run the command:

```
sbin/sysctl -p
```

to print `(-p)` the sysctl settings to the console. This will ensure ignoring redirects is enabled during reboot.

Also, with the arpspoofing, the server is acting as a router so if the server becomes bogged down with analyzing data other intensive CPU uses, then it can affect network performance. Also, if the arpspoof daemon is killed abruptly the subnet or the entire organization depending on where the server is placed could

And away we spoof!!!

potentially be incapacitated. Trust me on that one.

With port mirroring, the traffic flow is similar to that except the Eve is not acting as a router for the traffic because you can pull the network cable on that server and traffic continues to flow through the network uninterrupted. **DON'T DO THAT WITH ARPSPOOFING!!!!!! TRUST ME ON THAT!!!** However, since traffic is flowing through that mirrored port to Eve, traffic can still be manipulated. The rest of the paper will discuss that.

[<<< Previous](#)

[Home](#)

[Next >>>](#)

Dsniff 'n the Mirror

Essential preparation

Dsniff 'n the Mirror

[<<< Previous](#)

[Next >>>](#)

Essential preparation

1. GET PERMISSION FROM YOUR IMMEDIATE SUPERVISOR AND ALERT OTHER SYSTEM ADMINISTRATORS IN YOUR DEPARTMENT SO THEY CAN MONITOR FOR ANY ABNORMALITIES THAT MAY RESULT FROM THE USE OF THE PROGRAMS MENTIONED IN THIS PAPER!!!! THIS PAPER EXPLAINS HOW SENSITIVE INFORMATION CAN BE OBTAINED ON A NETWORK SO BE SURE YOU GET PERMISSION AND SECURE THE MONITORING SERVER !!!
2. ENABLE IP FORWARDING!!!
3. GET PERMISSION!!!
4. ENABLE IP FORWARDING!!!
5. GET PERMISSION!!!
6. ENABLE IP FORWARDING!!!
7. GET PERMISSION!!!
8. ENABLE IP FORWARDING!!!
9. GET PERMISSION!!!
10. GET PERMISSION!!!!!!!!!!!!!!
11. ONLY RUN SSH ON THE MONITORING SERVER, WITH ALL THE LATEST PATCHES, AND READ MY [OTHER PAPER](#) ON CREATING SECURE TUNNELS USING MINDTERM AND SSH.
12. A webserver will be used on **Eve** for the utilities used in this tutorial to view reports and graphs. If you are using Apache as the webserver edit your httpd.conf file and change the "BindAddress *" option to "BindAddress 127.0.0.1" and the "Listen" directive to "Listen 127.0.0.1:80". Therefore, you can only be on the local machine itself or you can create secure tunnels to the machine and view it remotely over a secure connection. Restart the webserver and use SSH to create a tunnel to **Eve** so that nothing is transmitted in clear text across the network and only users with legitimate accounts can access **Eve**. Typing:

And away we spoof!!!

```
/bin/netstat -anp --inet | grep httpd
```

should show the httpd daemon listening on the local interface:

```
tcp 0 0 127.0.0.1:80 0.0.0.0:* LISTEN 2676/httpd
```

To access **Eve** over ssh with the above settings, the following command using ssh will create a secure tunnel:

```
/usr/bin/ssh -l USERNAME -L 2080:127.0.0.1:80 Eve's ip address or hostname
```

The "-l" switch specifies the user to log in as (that is the letter "l") and "-L" means listen on the localhost port 2080 (**2080**) and forward to **Eve's** local interface

(**127.0.0.1**) on port 80 (**:80**). Then in a browser type:

```
http://localhost:2080
```

You will then be able to access **Eve** over a secure connection. If you are running other webservers do the same with the respective settings. Also, use an initial login page to access the webserver. **REMEMBER THAT THE INFORMATION THAT CAN BE OBTAINED FROM THE MONITORING SERVER CAN BE DISASTROUS IN THE WRONG HANDS.**

- Finally, GET PERMISSION!!!!!!

Software Used

- [RedHat](#) (dsniff works on other platforms. See the dsniff website for further information)
- [Dsniff](#)
- [Tcpdump](#)
- [Multi Router Traffic Grapher \(MRTG\)](#)
- [IPTRAF](#)
- [IP Flow Meter \(IPFM\)](#)
- [ccrypt](#)
- [Ngrep](#)
-

And away we spoof!!!

[NTOF](#)

•

[Snort](#)

•

[Snort-Report](#)

•

[Xinetd](#)

•

[Cryptcat](#)

•

[Fcheck](#)

[<<< Previous](#)

The Heart of the monitoring

[Home](#)

[Next >>>](#)

'To spoof or not to spoof, that is the packet'

Dsniff 'n the Mirror

[<<< Previous](#)

[Next >>>](#)

Ripped from the Headlines

There are a number of utilities that can be used to extract information from what is collected over the network while using the dsniff utilities, like tcpdump (or WinDump for windows), and also to capture information on the fly. Also, problems can be debugged by using these same utilities.

Below you can see how data can be extracted from was explained above using tcpdump. Normal viewing of tcpdump with the `-w` switch yields a bunch of garbage. However the command:

```
/usr/sbin/tcpdump -n -r /log/dump_"file name"
```

will output the contents of the file in human-readable format, well so to speak. The `-n` switch will output the data without doing a dns lookup so it is extracted much, much, much, much faster. `-r` specifies the data is read from a file. No data is collected from the network.

Or a particular host can be grepped from the file like this:

```
tcpdump -n -r /log/dump_"file name" | grep "host ip address"
```

If the file is large then a fast system should be used else system utilization will degrade until the process finishes. We have two hard core machines to do just that.

To view the ascii output of a particular host then tcpdump, tcpdump2ascii (<http://www.bogus.net/~codex/>) and sed will help. This command, though wicked, will do the job nicely:

```
/usr/sbin/tcpdump -n -x -r /log/dump_"file name" | tcpdump2ascii | sed -e './{H;$!d;}' -e 'x;/AAA/AA'
```

(where "AAA" is the name of the host you want to lookup)

It is a good idea to redirect the output into a file for analysis add "`filename.tcpdump`" to the end of the command above to do that.

```
/usr/sbin/tcpdump -n -x -r /log/dump_"file name" | tcpdump2ascii | sed -e './{H;$!d;}' -e 'x;/AAA/AA'
```

First the data is outputted into hexadecimal (`-x`) format from the tcpdump output file (`/usr/sbin/tcpdump -n -x -r /log/dump_"file name"`) then piped into tcpdump2ascii (`| tcpdump2ascii`). `tcpdump2ascii` converts the hexadecimal to ascii and each packet and its contents are separated by a paragraph, so sed is used to print each paragraph matching the keyword (`sed -e './{H;$!d;}' -e 'x;/AAA/AA'`) (in this sed command substitute AAA for the host name or anything for that matter). NOTE: The data is still kind of difficult to read because of the html format so another sed command will strip out most of the html code:

```
sed -e :a -e 's/<[^>]*>///g;///ba'
```

like so:

```
/usr/sbin/tcpdump -n -x -r /log/dump_"file name" | tcpdump2ascii | sed -e './{H;$!d;}' -e 'x;/AAA/AA'
```

The sed commands are taken from the "Sed One-liners" documentation found on various websites. Programs like [tcpslice](#) or [pcapmerge](#) can be used to extract particular data from tcpdump output to help narrow down the file size.

Here is a [sample tcpdump file](#) to test out the commands above. (Note: The first few packets is not a SYN scan. The frames and ads on securityfocus.com probably caused those multiple SYN connections to be created.)

This was shown just to show the flexibility of the free tools at your disposal. The bpf filters will be much more useful and handy as will ngrep.

Ngrep

Another excellent utility to use is ngrep (<http://ngrep.sourceforge.net/>). With ngrep certain texts can be sniffed from the network and displayed when a match is found. For example, to dump packets that contain the words, "**playboy, bunnies, bunny, Hefner**", this filter would output all packets that match including the local host host name and remote server name:

```
ngrep 'playboy|bunnies|bunny|Hefner'
```

or

```
ngrep 'playboy|bunn*|Hefner'
```

The pipes (|) delimit each key word. The words "**bunnies**" and "**bunny**" was substituted with the wildcard match (*). BPF filters can be added to the end of the text expressions like the above examples with the dsniff utilities and tcpdump. To sniff a particular host on the network this command will work:

```
ngrep '*' host "host ip or name"
```

Again redirecting the output to a file will make the output more readable and using the sed command above to strip out the html code will help as well. The documentation contains more examples of how to specify filters using ngrep. Instead of using the tcpdump extraction commands above, ngrep will read tcpdump output as well. For example, this command will do the same as the tcpdump output above but quite a bit prettier:

```
/usr/bin/ngrep -I /log/dump_"file name"
```

The "**-I**" switch implies data will be read from a file instead of from the network. This command will only print packets from the file that contain data. For more verbose output, including empty packets, and flags add a few extra switches:

```
/usr/bin/ngrep -qte -I /log/dump_"file name"
```

This command means be quiet ("**-q**"), print the timestamps (**t**), and show empty packets (**e**). If there were multiple interfaces in the file the "**-d**" switch means listen on that interface.

```
/usr/bin/ngrep -qtcd -I /log/dump_"file name"
```

Using the bpf filters, specific data can be extracted from a tcpdump file as well. Test out ngrep using the [sample tcpdump file](#) above.

Again, these tool should be used with care as very sensitive information can be obtained from it.

Snort

Snort is a superb IDS (Intrusion Detection System) that helps to identify probes and other anomalies with your network. It uses a database of signatures to detect known and potentially unknown attacks. It uses a new IDS search feature called heuristics. With this snort can detect attacks based on a general signature than the exact signature. If this is a known attack:

```
http://www.server.com/cgi-bin/script.cgi?file=../../../../etc/passwd
```

then snort uses the signature:

```
"/etc/passwd"
```

Therefore if this attack comes out:

And away we spoof!!!

```
http://www.server.com/cgi-bin/newattack.pl?page=/etc/passwd%00
```

then snort would catch it because it contains **"/etc/passwd"**.

See Dave Wreski's and Christopher Pallack's article on installing and configuring snort. (http://www.linuxsecurity.com/feature_stories/using-snort.html). Snort also has support for the unicode attacks that are quite prevalent with the still circulating Code-Red and Nimbda attacks. Again, since a directory traversal can contain a number of combinations using the unicode format just searching for the string "cmd.exe" is sufficient for snort to catch it.

Snort can also dump the actual exploit attempt for later review and analysis. It is not advised to do this on Eve because it can tax the CPU. Snort can read libpcap files so you can install it on another machine and parse the file.

```
/usr/sbin/snort -c /usr/snort/snort.conf -b -l /tmp/ -r snort.test -s
```

Use the snort config file located in **/usr/snort/** (**-c /usr/snort/snort.conf**), log the packets in tcpdump format (**-b**) in **/tmp** (**-l /tmp**) and read from the libpcap file snort.test (**-r snort.test**) and log to syslog (**-s**). Why log to syslog? Many snort reporting add-ons use the messages logged to syslog for analysis, like snort-report to process the snort logs and put them into a purty html or text format for web analysis or to email. Snort-report can create both text and html messages from the snort logs logged to syslog. It can be read while you are sipping that coffee in the morning or sent to you periodically through cron or its own report mailing utility, snort-rep-mail (You'll have to download the MIME::Lite perl module from CPAN). To create an html report of snort-report run this command:

```
/usr/local/bin/snort-rep --html /var/log/messages.> `hostname`-snort-report-`date +%b-%e-%Y`.html
```

By default snort-rep will output the results to stdout so it is ideal to redirect the output into a file. In this example, snort report creates an html report (**--html** or **-H**) and points to **/var/log/messages**, then is redirected (**>**) into a file with Eve's hostname (**hostname**) and the report generator (**snort-report**) along with the current short-date, numerical day and long-year (**date +%b-%e-%Y.html**). It would look something like this. **eve-snort-report-Feb-14-2002.html**. To create a text document from snort-report run this command:

```
/usr/local/bin/snort-rep --text /var/log/messages.> `hostname`-snort-report-`date +%b-%e-%Y`.txt
```

The results are the same as above except that a text file (**--text** or **-t**) is created. It will look something like this: **eve-snort-report-Feb-14-2002.txt**.

[<<< Previous](#)

Bandwidth usage

[Home](#)

Dsniff 'n the Mirror

[Next >>>](#)

Security Considerations

[<<< Previous](#)

[Next >>>](#)

Security Considerations

It cannot be stressed enough that the data that the monitoring server can obtain from the network is significant. Passwords, email messages, instant messages, etc.. Extreme care and attention to the security of Eve should be taken into consideration. Again, any tutorial you read that mentions installing a monitoring server should include information on how to properly secure the box. Be sure the box is located in a secure location where access is restricted. Check it to be sure no extra cables, serial cables, network cables, etc are not attached to it that shouldn't be. If it can be kept in a locked but ventilated room then that would be even better.

Take precautions starting from system boot time. Password-protect the bios and disable booting from the floppy drive and cdrom. See your bios documentation for doing this. Password-protect the lilo single-user boot mode. Unprotected single user mode could potentially allow someone to bypass any security restrictions and gain root access. This could potentially allow someone to change the root password by editing the `/etc/passwd` file. This is done by editing the root entry in the file by removing the "x", then running the `passwd` program from the command line.

Some ways to prevent this is to password protect lilo if you want to boot into single user mode, by typing:

```
LILO: linux single
```

at the Lilo prompt or:

```
LILO: linux s
```

(In RedHat 7.0 and above you get a graphical lilo prompt. Options can still be passed to it by typing **Ctrl+x**. This will drop you to the commandline **LILLO:** prompt.)

First, be sure that your `/etc/lilo.conf` file is readable and writeable only by root:

```
#!/bin/ls -l /etc/lilo.conf
-rw-r--r--  1 root    root          298 Jan 27 13:44 /etc/lilo.conf
```

If it as you see above **CHANGE IT!!!** Else, a normal user could view the contents of the file. This is important because the password in the `lilo.conf` file is not encrypted. **REPEAT THE PASSWORD IN /etc/lilo.conf IS NOT ENCRYPTED!!!! DO NOT MAKE IT YOUR SYSTEM ROOT PASSWORD!!!**

Change it with `chmod`:

```
#!/bin/chmod 600 /etc/lilo.conf
```

check it:

```
/bin/ls -l /etc/lilo.conf
-rw-----  1 root    root          298 Jan 27 13:44 /etc/lilo.conf
```

Now in your `/etc/lilo.conf` you will need to add the **"restricted"** parameter and the **"password"** parameter at the bottom of your linux partition information.

`/etc/lilo.conf` looks something like this:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
```

And away we spoof!!!

```
timeout=50
message=/boot/message
linear
default=linux

image=/boot/vmlinuz-2.4.9-12
    label=linux
    read-only
    root=/dev/hda9
image=/boot/vmlinuz-2.4.2-2
    label=linux.old
    read-only
    root=/dev/hda9
```

Add the parameters like this:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
linear
default=linux

image=/boot/vmlinuz-2.4.9-12
    label=linux
    read-only
    root=/dev/hda9
restricted
password=a password
image=/boot/vmlinuz-2.4.2-2
    label=linux.old
    read-only
    root=/dev/hda9
```

The "**restricted**" and "**password**" parameter together will require a password if any options are specified at the lilo commandline. If "**restricted**" is not added then a password would be required if any commandline options to lilo are specified and if you want to boot the operating system into normal mode. **DON'T FORGET TO RUN: /sbin/lilo -v**. If you make any changes to `/etc/lilo.conf`, always run that command to determine if there are any errors. Else, when you reboot you are going to hope you have a rescue disk handy.

For some defense in depth then use the "**sulogin**" program. This will require the root password before a shell is allocated. The prompt looks like the one when a filesystem is corrupt and you need to do maintenance. A friend told me that. 8-) Add the following line to your `/etc/inittab` file:

```
~~:S:wait:/sbin/sulogin
```

Data Security

Encrypt data on Eve using the encrypted loopback implementation or other encryption software. I like `ccrypt` because it is fast and it is usable between multiple operating systems (Linux, Solaris, Windows, MacOS X, FreeBSD, AIX, Linux for Alpha, Linux for PowerPC, Linux for Sparc).

`Ccrypt` has a feature where if you are encrypting data and something causes the encrypting process to break, you may still be able to recover some of the data. More information is in the documentation that comes with `ccrypt`. For large files you may want to tar the files first and then encrypt it with `ccrypt`.

And away we spoof!!!

```
#!/bin/tar -czvf "filename".tar.gz "file-to-encrypt"
```

First, first create a new tar archive (**-c**) and gzip the file (**-z**) for tighter compression, in verbose mode (**-v**) and specify a filename for the new archive (**-f**), then specify the file (or directory) to tar and gzip. Since the file is being gzipped as well, the ".gz" extension **has** to be added to it.

Next, run **ccrypt** to encrypt the file. You can use **ccrypt** or **ccencrypt**. You will be prompted for a password.

```
#!/usr/local/bin/ccrypt "filename".tar.gz
```

When you encrypt the file it will append a ".cpt" extension to it.

```
#!/usr/local/bin/ccrypt "filename".tar.gz.cpt
```

You can also encrypt multiple files at once.

```
#!/usr/local/bin/ccencrypt *
```

The "*" encrypts all files within the current directory but will not traverse any subdirectories. To decrypt files run the command:

```
#!/usr/local/bin/ccrypt -d "filename".tar.gz.cpt
```

The "-d" switch means to decrypt the file. You will be prompted for the password and if successful the file will be decrypted. Also, the command "**ccdecrypt**" can be used in place of having to type the "-d" switch. If you only want to decrypt data to stdout then use the command "**ccat**":

```
#!/usr/local/bin/ccat "filename".tar.gz.cpt
```

after you type the password the file contents will be redirected to stdin or redirected to a file if needs be but the original file will remain encrypted.

Remote Access

If you need to access Eve remotely and you are using port mirroring, you will need to add a second network card into the server and give just that card an ip address. Since it will have a live connection be sure to lock down the box and encrypt the data. Only enable SSH and wrap it with TCPWrappers, Xinetd, or PAM. Xinetd is preferred over TCPWrappers since you can do time-based access.controls with those programs.

Create a file called `sshd` in root of `/etc/xinetd.d/` using an editor. Then add the following contents:

```
service ssh
{
    socket_type          = stream
    wait                = no
    user                 = root
    server               = /usr/sbin/sshd
    port                 = 22
    server_args          = -i
    only_from            = 192.168.0.3 192.168.0.5
    access_times         = 08:00-21:00
    log_on_success       =          = USERID PID HOST EXIT DURATION
    log_on_failure       = ATTEMPT USERID HOST RECORD
    disable              = no
}
```

#Specifies the name of the service to use. If it doesn't exist then add it to the `/etc/services` file. `service ssh`

```
#Stream indicates the service uses tcp. dgram would be used if the service used udp.
```

And away we spoof!!!

```
{
    socket_type = stream

#this service is multithreaded
wait = no

#the owner of the daemon.  If the daemon is running in a chroot environment then change to the ap
user = root

#Location and name of the server
server = /usr/sbin/sshd

#Service port number
port = 22

#Arguments passed to the server.  "-i" tells sshd it is running from inetd.
server_args = -i

#Space delimited list of allowed hosts to connect to the ssh server.  192.168.0.0 would match all
#in the 192.168.0.0/24 subnet.
only_from = 192.168.0.3 192.168.0.5

#Times the server can be accessed in the format HH:MM:SS.  Space delimited for time ranges
#ie. access_times = 08:00-21:00 22:00-23:00
access_times = 08:00-21:00

#Log the following syslog level messages.
log_on_success = USERID PID HOST EXIT DURATION
log_on_failure = ATTEMPT USERID HOST RECORD

#Whether to enable or disable this service when xinetd is started.
disable = no
}
```

NOTE: Be sure to check /var/log/messages for any errors starting Xinetd

For TCPWrappers, in the /etc/hosts.deny file add:

```
ALL : ALL
```

in the /etc/hosts.allow file add the ip addresses allowed to access the server:

```
sshd : 192.168.0.3, 192.168.0.5
```

Be sure to only allow access to SSH from your workstation and only those people who need access to the system. In your sshd_config file modify the line:

```
PermitRootLogins yes
```

to

```
PermitRootLogins no
```

This will force users to login over ssh as a normal user then su to root.

Then add the "**AllowUsers**" directive followed by a space delimited list of allowed users:

```
AllowUsers karol terri doug
```

Also add the "**DenyUsers**" directive followed by a space delimited list of system users not allowed to use SSH at all.

```
DenyUsers shutdown halt nobody
```

And away we spoof!!!

and so forth. then restart your ssh server:

```
/etc/rc.d/init.d/sshd restart
```

Why the ridiculous security? The data that can be captured for this server can contain sensitive data and you want to have a Fort Knox style monitoring server. A high-level of physical security should be added in the security plan of this server. **YOU HAVE BEEN WARNED!!!** If an attacker can get into this machine and starts sniffing or transferring files, you can go ahead and go into Incident Recovery mode. Pardon the negativity but that is how it is. Any tutorial you read about port mirroring should reinforce the issue about protecting the server.

Restricting PAM-style

To allow only certain users to su to root follow these steps:

Edit the `/etc/group` file and add a comma-delimited list of users to the "wheel" group:

```
wheel:x:10:root,ken,doug,tammi,cindy
```

If the group "wheel" doesn't exist then create just the group "wheel".

```
#!/usr/sbin/groupadd wheel
```

In the `/etc/pam.d/su` file uncomment the line:

```
#auth required /lib/security/pam_wheel.so use_uid
```

There are two entries for trusting the wheel group in the `/etc/pam.d/su` file, by default. Be sure to uncomment the line above and **NOT**:

```
#auth sufficient /lib/security/pam_wheel.so use_uid
```

This would allow the user to su into root or other accounts without a password. **NASTY!!!** Delete that line from the file. The order of PAM directives are very important. When the "sufficient" directive is met then all other requirements in that file aren't checked. From the above example the password checking routine is completely bypassed by one variable, that is why the "required" directive is set because all other requirements must then be met, as well.

For additional security, the following line can be added to the top of the `/etc/pam.d/su` file:

```
auth required /lib/security/pam_listfile.so onerr=fail item=user sense=allow file=/etc/security/suok
```

`/lib/security/pam_listfile.so` is the PAM module used to restrict access to su based on a list of users in a "file", `onerr=fail` will output any errors to syslog, `sense=allow`, allows only the users specified in the "suok" file. (**NOTE: sense=deny would deny users the ability to su into the users account specified in the "suok" file**), `file=/etc/security/suok` this is the file to read the list of users to allow (or deny) su access account

Then create a file called `/etc/security/suok` and add the following entry:

```
root
```

The line added to the `/etc/pam.d/su` file says, only the user(s) specified in the `/etc/security/suok` file are allowed to be passed to the su command. In other words, anyone can invoke the su command but only if they are going to su to root. You can't su to another users account unless they are specified in the suok file. **NOTE: Root is not restricted by these settings.**

The chosen are few

Restrict the number of logins allowed for all users by editing the `/etc/security/limits.conf` file by adding the line:

```
*          hard    maxlogins    2
```

All users (*) are allowed only (**hard**) a maximum (**maxlogins** of two (**2**) logins. Root is not affected by these limits.

Also, limit the number of terminals and virtual terminals that a user can log into by editing the `/etc/securetty` file. Remove all the **tty*** entries except for one or two. This will restrict the number of virtual consoles that can be used. At the console type **Alt+(Function keys)** to switch virtual consoles, e.g. **Alt+F1**, **Alt+F2**, etc.

Hand in the google jar

File integrity is an absolute need for good security. A file integrity program not only alerts you to what an intruder may have installed, delete, etc. but you also learn more about your own computer by learning what changes in the course of general system performance. Those who are curious research the various files and directories that change to understand why it changes and what makes it change. So you are deepening your knowledge 10-fold by learning the internals of the operating systems. Fcheck is an excellent file-integrity utility. It is written in perl and runs on basically any os that can run a perl interpreter. It can use various checksum programs like md5 or l6 to create a hash on the files and directories specified in its configuration file.

Untar the fcheck distribution and **cd** into the fcheck directory that is created. Copy the fcheck perl script and the fcheck.cfg configuration file to `/etc`.

```
tar -xzvf fcheck-version.xxxx.tar.gz
```

```
cd fcheck
```

```
cp fcheck fcheck.cfg
```

Edit the `/etc/fcheck` file and change the **\$config** variable to the location of the fcheck.cfg file.

```
$config="/etc/fcheck.cfg";
```

Edit the `fcheck.cfg` file to start creating the files and directories to check. Here is a sample:

```
Directory      = /bin
Directory      = /usr/local/sbin/
Directory      = /usr/local/bin/
Directory      = /usr/sbin/
Directory      = /lib/
Directory      = /
Directory      = /etc/
Directory      = /home/
Directory      = /sbin/
Directory      = /usr/
Exclusion       = /var/log/messages
Exclusion       = /tmp
DataBase       = /usr/local/adm/log.dbf
FileTyper      = /bin/file
TimeZone      = EST5EDT
```

And away we spoof!!!

```
$Signature = /sbin/md5
```

The `fcheck.cfg` and the distribution's `INSTALL` has great documentation in them. I'll just hit a few things here. It is a good idea to put the directories that shouldn't change at the top of the configuration file so that the report that is printed will show those directories first. If you or some other admin hasn't changed or upgraded any packages that belong in those directories then there is a problem. The other directories are important but the directories at the top in this example shouldn't change at all unless there was an upgrade. Adding a slash at the end of a directory will check the root of that directory and all subdirectories and their contents. Not adding a slash at the end will only check the root of that directory. The **Database** location should be writeable by the user running `fcheck`. Be careful with the root `"/` directory because if you add it like this `"/` then it will check your entire file system. I like to specify the directories to check so that the report is in a nice formatted output. Note, in the example above that the `/usr/` directory will check `/usr/local/bin` and `/usr/local/sbin/`. That is done for redundancy. If something changed in `/usr/local/bin/` and it was missed at the top of the report then the opportunity is there to see if later when you are looking at the `/usr` directory later on. You can exclude files and directories from the check with the **"Exclusion"** variable. Be careful when excluding because crackers know that the `/tmp` and the `/var/log/` directories are sometimes excluded from integrity checks. That is a prime place for them to install a rootkit or whatever else they want. You may have unnecessary files and longer reports but it is better safe than sorry. The documentation is quite thorough with `FCheck` so be sure to consult it closely.

After configuring the config file create a database:

```
#!/etc/fcheck -ac
```

The `-a` switch means do all files and directories listed in the config file automatically and the `-c` switch means to create a new database.

NOTE: The first database may generate errors indicating it cannot traverse the directory specified. Be sure the directory listed in the check exists and it isn't a pseudo filesystem like `/proc`. Also, `FCheck` may not allow checking an empty directory. If this occurs just create a `.temp` file in the directory.

```
#!/bin/touch /somedir/.temp
```

To perform the file integrity check run the command

```
#!/etc/fcheck -a
```

The command above for checking the database can be redirected into a file or sent to you via email.

```
#!/etc/fcheck -a > integrity-report-`date +%b-%e-%Y`
```

```
#!/etc/fcheck -a | mail "user to mail to"
```

Other considerations

If you are using port mirroring then any firewall rules you enable to restrict access via `iptables` or `ipchains` or other firewall programs won't affect traffic flowing through the server. However, if you are using `arpspoofing`, then firewall rules will affect the server because the server is acting as a router. With port mirroring the data is flowing via the raw sockets. Raw sockets are "virtual interfaces" on a unix system. They aren't part of the `TCP/IP` stack so traffic freely flows through them. There is a project called **"Divert Sockets"** (<http://www.anr.mncn.org/~divert/index.shtml>) that uses a kernel feature and `ipchains` for the 2.2x kernel series to enable firewalling with the raw sockets. `Ipchains` and `iptables` uses the `TCP/IP` stack to affect network packet, since `arpspoofing` is causing Eve to act as a router firewall rules will affect the network packets. Most of the programs mentioned in this paper are written to listen and affect traffic on the raw sockets level.

And away we spoof!!!

Eve should not be used for heavy processing of data because it can become bogged down from the amount of traffic, alone, running through it. Data should be sent to another server for heavy process analysis. You can use scp or cryptcat to transfer the data to another machine. If you had a file called "data.tcpdump" then use scp to copy the data to a remote machine:

```
/usr/bin/scp data.tcpdump username@server:/path_to_copy_data
```

Cryptcat can be used as well. It has the functionality of netcat but with encryption of the data being sent using twofish and is available for linux and windows. Before you compile cryptcat be sure to edit the netcat.c file and change the default password, **metallica**, to something else. Change:

```
char * crypt_key_f9 = "metallica";  
to
```

```
char * crypt_key_f9 = "newpassword";  
Okay...don't use "newpassword" either.
```

Be sure that you delete the source code directory after you have compiled cryptcat and copied the binary somewhere on the server. If you leave the source code directory then someone could look in it and get your password. One more thing, **DON'T MAKE IT YOUR ROOT PASSWORD!!!** The password should be the same on the server and the client.

To transfer the file data . tcpdump. On Eve, run the command:

```
/bin/tar -cf - data.tcpdump | /usr/bin/cryptcat -l -p 4000
```

Create a new tar archive (**-c**) and since we are transferring the data across the network we use stdout as the file name (**-**) because when it transfers to the remote system it will be named "**data.tcpdump**". Then pipe (**|**) the data into cryptcat and listen (**-l**) on port 4000 (**-p 4000**) for connections. If port 4000 is used on your system already you will get an error that it cannot bind to the port.

```
Can't grab 0.0.0.0:4000 with bind
```

We are compressing it for faster transfer. Then on the remote system run the command:

```
/usr/bin/cryptcat Eve's ipaddress 4000 | tar -xf -
```

This will connect to Eve (**/usr/bin/cryptcat Eve's ipaddress**) on port 4000 (**4000**) and the piped into tar and extracted (**-xf -**) on the local system with its original file name, "data . tcpdump".

[<<< Previous](#)

[Home](#)

[Next >>>](#)

Ripped from the Headlines

Conclusion

Dsniff 'n the Mirror

[<<< Previous](#)

Thanks

Special thanks to Dave Wreski and the Linuxsecurity.com team for posting these articles. Thanks to Bone, Chris, Cris, Charla, Chrissy, *Mr. David*, Bob, CFCC, Pfeiffer University, Adam, and mutsman for their continued support for all that I do.

Duane Dunston is a Computer Security Analyst at [STG Inc.](#) for the [National Climatic Data Center](#) in wonderful Asheville, NC. He received his B.A. and M.S. degrees from [Pfeiffer University](#) and he has his [GSEC](#) certification from [SANS](#). He writes poetry, just started photography, and hangs out at The Old Europe Cafe, Early Girl's eatery, and Bean Street Cafe and still wakes up every morning ready to go to work. **If anybody works at NBC who reads this tell Ann Curry he says, "Hello".**

[<<< Previous](#)

[Home](#)

TODO

Dsniff 'n the Mirror

[<<< Previous](#)

[Next >>>](#)

'To spoof or not to spoof, that is the packet'

First, let's look at how normal traffic works. Here is an illustration:

1. *Node A transmits a frame to Node C.*
2. *The switch will examine this frame and determine what the intended host is. It will then set up a connection between Node A and Node C so that they have a 'private' connection.*
3. *Node C will receive the frame and will examine the address. After determining that it is the intended host, it will process the frame further.*

Please note that the hosts still perform the examination of the destination address even though the switch 'guarantees' that they are the intended host...In general, when Node A wants to communicate with Node C on the network, it sends an ARP request. Node C will send an ARP reply which will include the MAC address. Even in a switched environment, this initial ARP request is sent in a broadcast manner. It is possible for Node B to craft and send an unsolicited, fake ARP reply to Node A. This fake ARP reply will specify that Node B has the MAC address of Node C. Node A will unwittingly send the traffic to Node B since it professes to have the intended MAC address. (Sipes [Why your switched network isn't secure](#)) [\[1\]](#)

This technique of Node B intercepting the frames destined for Node C is called, "arp spoofing"; hence, the name of the utility that is being used from the dsniff package, "arpspoof". For more detailed information on arp spoofing read Stepen Whalen's paper [Intro to Arp Spoofing](#). What's being done, using Sipes's example above and in this paper, is that the monitoring server is intercepting all traffic on **ON-2** and then sending it to **Trent**. Therefore, we are able to get an accurate analysis of what is going on with our network.

Notes

[\[1\]](#)

[<<< Previous](#)
Essential preparation

[Home](#)

[Next >>>](#)
Defenses

Dsniff 'n the Mirror

Abstract

This is a practical step by step guide showing how to use Dsniff, MRTG, IP Flow Meter, Tcpdump, NTOP, and Ngrep, and others. It also provides a discussion of how and why we should monitor network traffic.

Table of Contents

[Introduction](#)

[The Heart of the monitoring](#)

[Essential preparation](#)

['To spoof or not to spoof, that is the packet'](#)

[Defenses](#)

[And away we spoof!!!](#)

[Bandwidth Control](#)

[Bandwidth usage](#)

[Ripped from the Headlines](#)

[Security Considerations](#)

[Conclusion](#)

[TODO](#)

[Thanks](#)

Introduction

In order to properly understand how your network operates and to debug any problems with network congestion, and other network issues, network monitoring is essential. It is also important to be able to monitor your network for signs on an attempted attack or have logs to corroborate an attack. It helps to quickly find out if your local network is having a problem, a particular host, or if some hosts are using up an excessive amount of bandwidth. It can also be used to just provide a historical analysis of how the network is being used. This paper discusses tools used to monitor "Our Network," hereby called **ON-2**. Also, the monitoring server will be called "**Eve**" and the gateway, "**Trent**".

Why?

This began as personal proof that Dsniff has good uses just as Dug Song mentions on his website

"I wrote these tools with honest intentions – to audit my own network, and to demonstrate the insecurity of most network application protocols. Please do not abuse this software." (Song dsniff)

Professionally, this was done to better see what is going on with our netowrk. I used to work for an organization that had a restrictive commercial firewall and we weren't allowed access to the console and it doesn't have the utilities needed to accurately monitor the network. After some experimenting under the cover of pretending to do real work, this paper was the result. Just kidding. Thanks to my former employer [Cape Fear Community College](#), this paper is the result.

[Next >>>](#)

The Heart of the monitoring

Dsniff 'n the Mirror

[<<< Previous](#)

[Next >>>](#)

TODO

Play with some more monitoring tools.

[<< Previous](#)

Conclusion

[Home](#)

[Next >>](#)

Thanks